A FLIGHT EXPERT SYSTEM FOR ON-BOARD FAULT MONITORING AND DIAGNOSIS

Langley Grant NAG1-513

Final Report

NASA Grant – NAG1 513

October 31, 1990

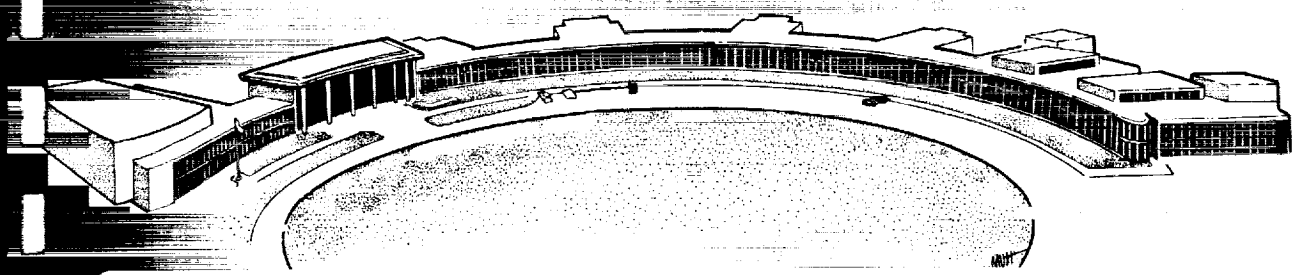
N91-13452 310159 p-52

Submitted by

Professor Moonis Ali
Principal Investigator
The University of Tennessee Space Institute
Tullahoma, TN 37388

Prepared for

NASA Langley Research Center
Hampton, VA 23665–5225
Attn: Dr. Kathy Abbott



THE UNIVERSITY of TENNESSEE

SPACE INSTITUTE

Tullahoma, Tennessee

A FLIGHT EXPERT SYSTEM FOR ON-BOARD FAULT MONITORING AND DIAGNOSIS

Final Report

NASA Grant – NAG1 513

October 31, 1990

Submitted by

Professor Moonis Ali
Principal Investigator
The University of Tennessee Space Institute
Tullahoma, TN 37388

Prepared for

NASA Langley Research Center
Hampton, VA 23665–5225
Attn: Dr. Kathy Abbott

TABLE OF CONTENTS

A. AN OVERVIEW

The increasing complexity of modern aircraft creates a need for a larger number of caution and warning devices. But more alerts require more memorization and higher work loads for the pilot and tend to induce a higher probability of errors. A solution to this problem is the use of an expert system for aiding in diagnosing and recovering from in-flight faults. By exploiting large amounts of domain knowledge, expert systems are able to accurately assess information and determine the cause of a fault. An on-board expert system would then help pilots focus on the source of a fault and increase the effectiveness of their attempts to correct faults. We developed an architecture for a flight expert system (FLES) to assist pilots in monitoring, diagnosing and recovering from inflight faults. A prototype of FLES has been implemented. The architecture and the prototype are described in Sections B of this report.

With the ever growing interest in the use of expert system technology for sophisticated problem solving, it becomes increasingly important to expedite the traditional knowledge acquisition process from which these systems evolve. The traditional process (in which a knowledge engineer familiarizes himself with the domain, helps the domain expert formalize the required knowledge, and finally implements the representation of this knowledge in machine usable form) is slow, cumbersome, and time-consuming. In the research reported in Section C, an attempt has been made to automate the knowledge acquisition process by employing a "learning by being told" methodology. The scope of acquired knowledge ranges from domain knowledge, including the information about objects and their relationships, to the procedural knowledge associated with the functionality of the mechanisms.

AKAS (Automatic Knowledge Acquisition System) is the constructed prototype for demonstrating proof of concept, in which the expert directly interfaces with the knowledge acquisition system to ultimately construct the knowledge base for the particular application. In this prototype, the expert talks directly to the system using a natural language restricted only by the extent of the definitions in an analyzer dictionary, i.e., the interface understands a subset of concepts related to a given domain. In this case, the domain is the electrical system of the BOEING 737. The system contains a set of seed concepts and vocabulary to capture the required information. Using this information, the system extracts and differentiates the information into two fundamental classes, i.e., the information relating to physical objects and their relationships to each other in the domain and

domain-dependent heuristics. These two classes of knowledge are represented in the form of frames and functions, respectively, to create the knowledge base and to augment the control strategy of the expert system.

In the research reported in Section D, efforts have been made to define and employ heuristics as well as algorithmic rules to conceptualize data produced by normal and faulty jet engine behavior examples. These rules have been employed in developing the machine learning system called MLS. The input to MLS is examples which contain data of normal and faulty engine behavior and which are obtained from an engine simulation program. This data includes sensor values in time, start and end points of the significant variations in the sensor values, and first and second derivatives of these variations. MLS first transforms the data into discrete selectors. Partial descriptions formed by those selectors are then generalized or specialized to generate concept descriptions about faults. The concepts are represented in the form of characteristic and discriminant descriptions, which are stored in the knowledge base and are employed to diagnose faults. MLS has been successfully tested on jet engine examples.

B. A FLIGHT EXPERT SYSTEM (FLES)

1. Introduction

The growing complexity of modern aircraft has led to an increase in the number of monitoring devices required to fly them. This situation has produced a higher workload for the flight crew and resulted in a greater percentage of accidents due to human error [1,2]. These errors are often made in situations where multiple faults have occurred and a confusing assortment of lights and warning devices are activated. Under such circumstances, it is difficult for a pilot to quickly isolate the cause of a fault from all of its symptoms.

One possible solution to this problem is the use of digital avionics in combination with an expert system which performs fault diagnosis automatically. By replacing the electromechanical gauges, switches and knobs of contemporary cockpits with color CRTs, head-up displays, touch-sensitive panels and voice avionics, flight crews can be presented with data in a more comprehensible format [3-5]. More importantly, unessential information need not be given, and whatever data is displayed can be prioritized so that critical information appears in the most conspicuous location.

Recently, several researchers have attempted to define the issues involved in building a flight expert system, but few have actually implemented such a system. Anderson et al. [6] proposed a rule-based system with an emphasis towards automating emergency procedures, and have developed a prototype to test their ideas. Cross [7] argues that traditional systems (e.g. rule-based) will not be sufficient without some capacity for performing common sense reasoning. In Klos et al. [8], a layered approach is suggested with respect to the level of automation desirable in a flight expert system. While we recognize the usefulness of defining the issues involved in implementing a flight expert system, it would seem that they are best discovered by experience. Thus, our approach has been to develop a working prototype and learn the issues in the process.

From the beginning, our efforts have been directed by knowledge collected when interviewing the pilots and maintenance personnel who served as our domain experts. Based on our discussions with these experts, we identified the following criteria for our initial prototype. First, because the sensors themselves are sometimes more unreliable than the airplane components, sensor data are always verified by using available global information. Second, the processing of faults is prioritized so that dangerous faults are checked for be-

fore those with less impact on the performance of the aircraft. Similarly, faults that are highly unlikely to occur are not checked for until after more likely possibilities are eliminated. Third, distinctions are made between fault causes and fault symptoms. Fourth, an accurate model of the status of the aircraft is maintained at all times. The model includes beliefs regarding which components are currently inoperative and the justifications for those beliefs. This model can be used by the explanation and diagnostic functions.

2. System Architecture

There are several classes of subsystems in FLES. In this report we have concentrated our description on two classes: the flight monitor and the fault interrupt controller. The flight monitor subsystems monitor sensors associated with pitch, yaw, roll and all other attributes in need of regular attention and/or adjustment. Because the values of these attributes may vary according to each phase of flight (i.e. taxi, takeoff, etc), a different subsystem is used for monitoring each phase. When a top-level subsystem of the flight monitor detects abnormalities, it verifies each one and issues an interrupt to invoke a monitor interrupt analyzer. This analyzer performs additional verification and seeks to diagnose them from a global perspective.

In addition to monitor interrupts, there are four other categories of interrupts built into the design of FLES, i.e., 1) electrical systems fault interrupt, 2) hydraulic system fault interrupt, 3) engine fault interrupt and 4) fire interrupt. Interrupts in each category are generated when sensors monitoring aircraft components detect values outside of their normal operating range. The corresponding interrupt analyzer then constructs hypotheses regarding the components in question. If there is sufficient supporting evidence, the hypothesis is verified and passed to a diagnostic routine. At this point it is determined if the inoperative component is the source of the problem or the side effect of some other fault. The results are then stored in a dynamically maintained non-monotonic knowledge base.

3. Fault Classes

In developing FLES, we found it helpful to examine the characteristics of faults, and to identify what types of faults occur in the flight domain. Early on, we recognized that some faults are due to improperly adjusted controls. Typical examples of this are extended flaps during flight, insufficient throttle during takeoff or excessive airspeed on approach. We refer to this type of fault as a *maladjustment.* A second type of fault, a *malfunction,*

exists when a component is no longer able to perform its intended task. Examples of malfunctions include electrical shorts, hydraulic fluid leaks and engine flame-outs.

Another observation was of the relationship between faults and the times at which they occur. We have noticed that maladjustments are sensitive to the phase of flight that the aircraft is in. The flap positions, for example, vary depending on whether the plane is trying to takeoff, land, cruise, etc. This means that to detect a maladjustment in the flaps the system must be aware of their current setting and the current flight phase. We have found this to be true for each type of maladjustment.

This is in sharp contrast to malfunctions, which are largely independent of the flight phase. A short in an electrical bus, for example, is obviously a fault at any time. Although there are exceptions to this independence, (e.g., the degree of danger associated with each malfunction may vary considerably from phase to phase), it has been useful to maintain the distinction between malfunctions and maladjustments. This distinction had a significant impact on the architecture of FLES.

4. Knowledge Representation

The success of an expert system largely depends on its use of large quantities of domain knowledge to guide the problem solving task. In FLES, domain knowledge is represented in a network of frames. The representational power of frames has been described elsewhere in the literature [9] and will not be repeated here. In this report we suffice to say that a frame may be viewed as a collection of related information about some object. FLES currently maintains three types of frames: sensor, malfunction and maladjustment.

Each sensor is associated with one sensor frame, which stores all important information regarding the sensor. Sensor frames contain the following items:

- *Range:* a safe range of values for the sensor.
- *Current Value:* the current value of the sensor.
- *Associated Fault:* a pointer to a malfunction or maladjustment frame, depending on which type the sensor was monitoring. For example, a sensor monitoring hydraulic fluid quantity would have a pointer to the hydraulic reservoir malfunction frame.

Knowledge about malfunctions is represented in malfunction frames. Malfunction

frames contain the following:

- *Type:* this refers to which subsystem the malfunction is in. Possible entries are hydraulic, pneumatic, engine, fuel or electrical.
- *Sensors:* a list of sensors responsible for detecting the malfunction. Where possible, we have eliminated the need for redundant sensors by exploiting causal relationships between faults. Thus, rather than verify a conclusion by way of voting between identical sensors we do so by observing the presence or absence of expected side effects.
- *Status:* this represents the status of the malfunction. If the status is **on**, the component is functioning properly and no malfunction exists. If the status is **off**, the component is inoperative as a result of some other malfunction. A status of **bad** is assigned when the component has itself malfunctioned.
- *Degree of Danger:* this is a list of numerical values meant to represent the danger associated with the malfunction in each flight phase. The values are derived from opinions given by pilots and are not meant to be precise. Rather, they serve as estimates that allow us to dintinguish extremely dangerous faults from those of less significance.
- *Feasibility:* a numerical value that estimates the feasibility of the malfunction actually occurring. Like the degree of danger, these values were obtained from pilots. This value is used to focus attention on likely faults before checking for those that are less likely to occur.
- *Possible Causes:* a list of faults that would cause this malfunction to occur. The malfunction frame for a hydraulic pump, for example, would contain entries for a loss of hydraulic fluid (i.e., a leak) and a loss of electrical power, since either would cause a drop in pressure at the pump. In the first case, the pump would start cavitating once the quantity dropped sufficiently. In the second, it would quit pumping until electricity was restored to it. Boolean combinations of specific conditions may be used to represent a possible cause.
- *Side Effects:* this is a list of side effects (i.e., other faults) that should be present if this malfunction exists. Note that a side effect can be a malfunction or a maladjustment. A loss of hydraulic pump pressure is one side effect of a hydraulic leak.

Maladjustment frames are used to represent domain knowledge pertaining to specific flight characteristics such as throttle setting, flap positions, pitch, yaw, airspeed, etc. The entries for maladjustments are identical to those of malfunctions with the following

exceptions:

- *Type:* not used.
- *Sensors:* a list of sensors monitoring this characteristic. Unlike malfunctions, it is sometimes desirable to employ a small set of redundant sensors to detect maladjustments. The reason for this is that the characteristics monitored by these sensors play a larger role in determining the overall status of the plane (e.g., whether or not it is on course), and so knowing their precise values is important. This contrast is illustrated in the following example.

Hydraulic quantity is monitored by a single sensor (figure 1). If the sensor was defective and failed to detect a loss of quantity, the fault would still be detected by the hydraulic pressure sensors. Since there are several of these (e.g., one for each of two pumps and one for the overall system pressure) and since they would all record a simultaneous drop in pressure, we could still conclude that the hydraulic quantity was low. Precisely how low would not be known, but that would not prevent us from isolating the source of the fault and determining a proper corrective action. On the other hand, if we only had one sensor to monitor an attribute such as pitch we would be severely hindered by a failure in that sensor. For although changes in pitch can be detected by corresponding changes in airspeed and altitude, it is important to know the exact amounts that each of these has changed.

- *Possible Causes:* this contains a list of possible causes for the maladjustment to have occurred. Each possible cause is represented by a list of specific sensors to test. The maladjustment frame for a bank condition has six possible causes: pilot or autopilot, control surfaces, fuel imbalance, power imbalance, turbulence and open gear doors. The first possibility, for example, is a list of sensors to check in determining whether or not the bank was caused by the pilot's actions.

5. Fault Diagnosis

The distinction between maladjustments and malfunctions has had a pronounced effect on the architecture of FLES. In order to appropriately service both types of faults, two diagnostic subsystems have been developed: the sensor interrupt handler and the flight phase monitor.

Malfunctions are detected and diagnosed by the sensor interrupt handler (SIH)[13-19]. The SIH architecture contains a top-level monitor as well as separate subsystems for pneumatic, hydraulic, fuel, engine and electrical malfunctions. Because malfunctions do not frequently occur, each SIH subsystem remains dormant until a fault of that type is detected. At that time an interrupt is sent to the top-level monitor. The top-level monitor constructs a hypothesis regarding the component in question (i.e., that it is inoperative) and activates the appropriate SIH subsystem. The SIH subsystem then verifies that the component is actually inoperative and, if so, determines whether it is the problem source or the symptom of another malfunction.

The flight phase monitor is a continuously operating subsystem that regularly checks critical flight characteristics such as yaw, pitch, roll and airspeed by testing a predefined set of sensors. Because the relevant set of sensors and the proper range of values for each one change according to the current phase of flight, FLES architecture provides a separate subsystem for monitoring taxi, takeoff, climb, cruise, descend and landing flight phases. For each phase, a specific set of sensor values is regularly compared with their proper range. In the event that one is out of range a hypothesis is constructed regarding that sensor and the attribute it was monitoring. The details of the control strategy are given in the next section.

6. Control Strategy

The heuristic-based control strategy employed by FLES is based on the control mechanism used by domain experts. Several domain experts were given diagnosis problems and their problem solving techniques were observed. We found that these experts, when given a set of sensors whose values were out of range, created a set of hypotheses. These hypotheses were considered according to what seemed most probable and what had the greatest potential for danger. Then they employed causal relationships and global system knowledge to confirm or reject hypotheses, as well as to differentiate the root causes from side effects.

In FLES, when sensor values leave their normal operating range, interrupts are raised, causing hypotheses regarding possible faults to be constructed. A hypothesis contains the name of the component monitored by the active sensor (when a sensor raises an interrupt because it has left its normal operating limits, we refer to it as active), a list of all the

sensors that detected the component failure (some components are monitored by several sensors) and a priority for the hypothesis which is based on the degree of danger and the likelihood values of the fault frame for the component. When a sensor raises an interrupt and finds that its hypothesis already exists, we add the sensor to the existing hypothesis rather than create a new one.

Once a hypothesis is constructed, it is placed in a hypothesis bulletin board (HPBB). If there are other hypotheses already in the HPBB, it is placed according to the priority assigned to it. Like Hearsay's blackboard [10], the HPBB is used as a depository for hypotheses from differing knowledge sources (i.e., the sensor interrupts analyzers). It is different from Hearsay, however, in that the hypotheses are not mutually exclusive. Because they only represent a belief that a component is inoperative, rather than a belief about which component is the source of the fault, it is possible for all of the hypotheses placed in the HPBB to be verified. An additional distinction comes from the fact that hypotheses are processed outside of the HPBB, i.e., once they are removed for verification, they are not put back into the HPBB regardless of the outcome of the verification process.

The conclusions of the verification procedure are stored in the deduced knowledge base (DKB). Here, once a component is believed to be inoperative it is recorded as such, along with the sensors upon which the belief is based. We have implemented a modified version of Doyle's Truth Maintenance System [11] to provide the mechanism for storing beliefs and their justifications. In this system, belief that a component is inoperative is based on the belief that its sensors are out of their proper range of values. If the sensor returns to its proper range (e.g., as would happen if a stalled engine were restarted), the change in status of the sensor is propagated to the component to change its status, as well.

Each entry in the DKB is assigned a reference number, which is displayed to the pilot along with the belief. By mentioning this number, the pilot can request an explanation for the belief (which is generated from its justification). A certainty factor, computed using Mycin's approach for accumulating evidence [12], is also included in each entry in the DKB. The certainty factor is computed on the basis of the number of sensors in the justification.

In FLES, after an interrupt analyzer receives an interrupt, it constructs and adds hypotheses to the HPBB. Because a single fault may have several side effects (which may

be detected by many different sensors), several hypotheses may be entered into the HPBB. If, for example, generator #1 (shown in Fig. 2) failed, several components would lose power, causing the following hypotheses to be constructed and placed in the HPBB:

1. (generator #1 40 gen#1-freq, gen#1-ac-volt)
2. (vscf #1 35 vscf#1-freq, vscf#1-ac-volt)
3. (ac bus #1 30 acbus#1-volt, acbus#1-curr)
4. (rectifier #1 25 rect#1-dc-volt, rect#1-dc-curr)

etc.

The structures and examples of the sensor and fault frames referred in the hypotheses above are shown in Fig. 3 and Fig. 4. The hypotheses in the HPBB are ordered according to their priority, which is computed on the basis of the component's degree of danger and likelihood. It would be useful here for the system to recognize when a burst of interrupts is due to a single failure and associate all of the resulting hypotheses together in a manner that reflects that realization (and thus in this example, no ordering would be necessary because all of the hypotheses shown could be serviced as a single unit). Indeed, in the example given, this would appear to be a simple task. Unfortunately, with more complex examples, we find that when a hypothesis is created, there are many faults (or combinations of faults) that it could be attributed to. Therefore, in the current prototype no attempt is made to associate hypotheses with one another within the HPBB; rather, this task is performed later by the diagnosis procedure.

Once FLES has serviced each interrupt by constructing the necessary hypotheses, a verification procedure is invoked. This is done to prevent a single defective sensor from issuing an interrupt and causing the system to believe faults exist where they do not. The verification procedure removes hypotheses from the HPBB, one at a time, and examines the amount of evidence in support of each one. Hypotheses supported by two or more sensors are automatically validated. When there is only one sensor in support of the hypothesis, the parents and children of the named fault are checked. If, by observing the parents, a likely cause of the fault can be found, the verification succeeds. Alternatively, if the expected side effects of the fault are seen in the children (i.e., if they, too, are inoperative), the verification will succeed. In circumstances where no verification can be found, the hypothesis is discarded. As previously mentioned the results of each successful verification are recorded in the deduced knowledge base.

Once the verification procedure confirms that a component is inoperative, a diagnostic procedure is invoked to determine whether the component is the source of the fault or the side effect of some other fault. The diagnosis is based on the model of structure given by the parent and children slots of the fault frames. If, for example, a hypothesis of the form

(vscf #1 35 vscf#1-freq, vscf#1-ac-volt)

was being diagnosed, the parent slot of the vscf #1 frame would be examined. This slot has a value of "generator #1" (figure 4). Thus, if the generator is found to be inoperative, the vscf would be diagnosed as "off". If, however, the generator was functioning properly, the vscf would be assigned a status of "bad". The nature of this diagnostic process is non-monotonic in that in the absence of any information claiming that the generator has failed, the system concludes that the vscf is the source of the fault and labels its status as "bad". If, however, it later finds that the generator is indeed inoperative, the status of the vscf is changed from 'bad' to 'off'. The outcome of the diagnosis is always recorded in the status slot of the component's fault frame.

7. Conclusion

The implementation of a system like FLES is an evolutionary process, requiring the ability to experiment with various possibilities and to easily modify or adapt the program to test new ideas. Demonstrations of the prototype have been favorably received by domain experts; however, there are many directions in which this work can be expanded. Currently, a sensor value oscillating in and out of its range causes a serious problem by adding multiple copies of the same hypotheses at a rate faster than the system can handle. To correct this problem, we will extend (in our future research) the system to record slowly developing faults through the deteriorating sensor values before they reach the critical range.

Another area in which we will be working is related to the assessment of fault impact. In a given situation it may be life-saving if the system could assess and inform the pilot about the capabilities and incapabilities of airplane components in the presence of a single or multiple fault occurrence.

In the majority of diagnostic processes, faults are detected, analyzed and recorded without reference to any temporal knowledge. In some diagnostic processes, however, we realized that it was important to determine the time periods in which faults occurred. For example, the diagnostic strategy employed when two generators have failed could be

altered by knowledge of whether or not they failed simultaneously (and therefore appear to be related). For this reason, we would like to include the use of temporal knowledge and employ methods representing and reasoning about temporal knowledge globally. With a few notable exceptions [22-26] there has been little work done on developing and employing such methods. We believe further research on this topic will enhance the capabilities of FLES significantly. The robustness of diagnostic systems, designed for solving problems of complex physical systems, is another area which must be focussed [27–28].

C. ACQUISITION OF DOMAIN KNOWLEDGE FROM HUMAN EXPERTS

1. Introduction

With the ever growing interest in the use of expert system technology for sophisticated problem-solving, it becomes increasingly more important to expedite the traditional knowledge acquisition process from which these systems evolve. The traditional process (in which a knowledge engineer familiarizes himself with the domain, helps the domain expert formalize the required knowledge, and finally implements the representation of this knowledge in machine usable form) can be characterized as slow, cumbersome and time-consuming. Of course, the reason for this characterization is the initial mismatch in domain expertise between the domain expert and the knowledge engineer. Edward Fiegenbaum [31] calls this mismatch the " knowledge acquisition bottleneck. " This bottleneck can occur not only because of a mismatch in expertise (not understanding the vocabulary and fundamental concepts of the domain), but also because it can often be difficult for a domain expert to precisely describe, step by step, a given problem-solving strategy (a domain heuristic). Often a domain expert may not describe many solution steps because they seem intuitively obvious. However, before a computer can be completely successful in problem-solving, all solution steps must be describable and understood to be implementable.

The significance of this "bottleneck" in understanding the domain knowledge is that this often prolongs the knowledge acquisition process. With the lengthening of this process often come problems related to manpower availability, scheduling, and delays in system development, possibly leading to premature termination of the project. If the person acting as expert is only a part-time expert then he may not always be available when needed by the knowledge engineer. In trying to arrange interview meetings, there may be scheduling conflicts between the expert and the knowledge engineer, thus delaying the construction of the system. If the elapsed development time becomes protracted, then management, who originally sanctioned the construction of the system, may become disenchanted with the rate of development and cancel the system.

Realizing the importance of trying to capture both domain and heuristic knowledge, researchers have constantly strived to enhance this process through automation. One very popular automation approach is the use of a learning paradigm called "learning by being told." This approach has been used in such systems as TEIRESIAS by Randall Davis [30],

KAS by Richard Duda [34] and NANOKLAUS by Norman Haas and Gary G. Hendrix [32], as well as the system described in this paper, AKAS (Automatic Knowledge Acquisition System).

Although these have used the same general learning approach, the objectives of each of them have varied. TEIRESIAS, using meta-knowledge and a restricted natural language vocabulary, attempted to capture rule-like knowledge for the medical diagnostic system, MYCIN. KAS was essentially the PROSPECTOR program without its domain-specific knowledge. Its objective was to allow application of the PROSPECTOR control strategy, "a consultation program developed for problem diagnosis in mineral exploration," [34] on new problems representable in the KAS rule language. The objective of NANOKLAUS and the KLAUS-based systems, in general, "was to collect and organize aggregations of individual facts for use in question-answering tasks." [32] The objective of the research creating AKAS was to build a knowledge acquisition tool for FLES [13–19], a diagnostic flight expert system project originated at UTSI. It was the intent of AKAS to try to capture, through natural language interaction with the domain expert, information about domain objects (e.g, generators, csds, busses, etc.), to be expressed in the form of frames and slot values, and specific problem-solving strategies (heuristics), to be expressed in the form of Lisp functions (because FLES was not a rule-based system employing forward or backward chaining but frame-based using specific functions to perform specific domain tasks, such as fault detection, fault/sensor verification and fault diagnosis.) The sections which follow discuss the system's development domain, overview the AKAS architecture, discuss the module implementations using two examples illustrating the acquisition of both domain and heuristic knowledge, and describe possible future research.

2. The AKAS Development Domain

Because of previous experience with the development of a Boeing 737 electrical system knowledge base for FLES [18], the electrical domain was chosen for use in developing the AKAS prototype. Based on conversations with domain experts, domain concepts, domain relationships, and diagnostic heuristics were identified. The information gathered during this manual acquisition process became the basis of the seed information incorporated into developing the automatic system.

3. System Architecture Overview

AKAS consists of four basic modules as illustrated in Figure 5. They are 1) the system administration module (SAM), 2) the concept comprehension module (CCM), 3) the knowledge classification/extraction module (KCEM) and 4) the knowledge incorporation module (KIM). The target knowledge base and expert system control strategy boxes are intended to represent the repository of the information generated by the system. The box labeled "explanation capabilities" represents the ability of the system to explain the generated objects in relation to the original input sentences. The explanation facilities will be discussed in more detail in a later section of the paper.

The system administration module performs the system's bookkeeping tasks, such as 1) identifying the expert, 2) allowing the user to set/adjust his/her help level, 3) correlating the input with created structures and functions, 4) relating the created information to the expert source and 5) saving the information for later use.

The concept comprehension module provides the natural language interface to the user and is the workhorse of the acquisition system. The control strategy used in this module is an adapted version of the analyzer designed and written by Richard E. Cullingford [29] and his colleagues. The system was adapted from its original Franz Lisp to Zetalisp for operation on the Symbolics 3670. Using a vocabulary designed for the electrical domain, as previously described, it was possible to create conceptual-dependency-like structures to capture the user/expert information. The output of this module becomes the input for the knowledge classification/extraction module.

The knowledge classification/extraction module classifies the input into one of two fundamental categories–domain objects and relationships, or domain-dependent heuristic knowledge and domain-related concepts–and generates the appropriate information as input to the knowledge incorporation module. Classification is based upon the primitives created for the concept comprehension module. For example, generator, csd, and bus are physical domain objects and connect or power are verbs which may indicate relationships between two objects. For domain heuristics, such words as "determine", "diagnose", "detect" or "verify" might be used to describe how some operation is performed. An example of a domain-related concept would be some specific concept or object, such as supporting sensor or bulletin board.

The knowledge incorporation module transforms the extracted information into the appropriate LISP forms, either LISP objects (i.e., internal machine representations, not flavor objects) or LISP functions. The LISP objects, in this case, are frames which represent the physical objects in the domain. The object interrelationships in the domain are represented by the slot values. The LISP functions represent the domain-dependent heuristic information. This module produces the final output forms to be used in either the target knowledge base or the target control strategy.

In developing the overall design of AKAS, research problems, mentioned primarily in the design of KLAUS (Knowledge Learning and Using Systems) [32], were seriously considered, which explains the presence of some system features and some elements of the system design. Briefly, the KLAUS philosophy advocates 1) the use of interactive, mixed-initiative dialogues, 2) a powerful natural language processing capability, 3) a structure for lexical entries to acquire new lexical information, 4) identification of seed concepts and seed vocabulary for inclusion in a core system, 5) a methodology for integrating new concepts into a system's knowledge base, and 6) a means for detecting and dealing with inconsistent data. The KLAUS design philosophy includes other points. The ones mentioned above are included in the design of AKAS.

4. System Implementation

AKAS has been implemented on a Symbolics 3670 using the defsystem and defpackage facilities available under Genera 7.1. The current user interface was designed using the Frame-up facility, a utility also available under Genera 7.1. The system is divided into two fundamental packages AKAS, implemented in Common Lisp and NLP, implemented in Zetalisp. (Common Lisp was not available on the Symbolics at the time of the NLP conversion.) The AKAS package contains the functions created for the implementation of the System Administration module, the Knowledge Classification/Extraction module, and the Knowledge Incorporation module. The NLP package contains the control strategy for the implementation of the Concept Comprehension module. It also contains the seed vocabulary created for the purposes of demonstrating the overall AKAS system.

Explaining AKAS's overall control strategy is best accomplished through the use of two examples. One will relate to domain objects and relationships. The other will describe a portion of a domain heuristic related to verifying a fault in the diagnostic strategy. Each

major module will be explained using these two examples.

4.1 The System Administration Module

The first module encountered by the user, SAM, requires that the user/expert login before proceeding. This permits the system to know the user's/expert's name to accomplish three things: 1) check the user table to see if the user has previous experience with the system, 2) allow creation of an expert flavor associated with the expert for later reference, and 3) create a file named for the expert where all system-generated information will be stored. After a brief message about the system and how to get help, the system begins with the prompt, "Enter Sentence:." It is at this point that the concept comprehension module begins its work. As long as the session continues, the entry point for the user will be the "Enter Sentence:" prompt. The user no longer must identify himself until the next login.

4.2 The Concept Comprehension Module

The second module in the system design is the Concept Comprehension module. As discussed earlier in the system overview, this module is the workhorse of AKAS providing the natural language interface to the user/expert. Understanding this module's design philosophy and control strategy will enhance understanding the criticality of the analyzer dictionary and its role in this module's operation.

The CCM Design Philosophy

The basic philosophy of the concept comprehension module was centered around the idea of "conceptual analysis or the mapping of natural language sentences into a semantic (meaning) representation. Thus, the program is technically a surface-semantic conceptual analyzer whose main job is to convert a surface form, an object encoding an English sentence or sentence fragment, into a meaning representation at the level of surface semantic categories or ERKS". [29] The ERKS, ("an amalgam of ideas from Conceptual Dependency, Commonsense Algorithms, and Preference Semantics") [29], is the method used to represent concepts to be found in the user-defined system. In conjunction with the ERKS, two other fundamental tools are used; these are the use of wordsense definitions (different uses of the same word, such as take - in take a pill (ingest) or take a train (ptrans)), and the analyzer dictionary definitions. The analyzer dictionary definitions provide the overall

driving force in the control strategy.

The CCM Control Strategy

To accomplish the conversion process from surface form into meaning representation, the control functions of the program focus on the lexicon as the locus of processing. That is, the analyzer dictionary definitions directly control the flow of processing. The overall design of the analyzer definitions takes its form from the production system, that is the definitions are in the form of test-action pairs. Thus in this approach there is no explicit grammar as in an ATN. This lack of a specific grammar allows processing of sentence fragments, ungrammatical inputs, as well as well-formed sentences.

The focus on the lexicon as the locus of processing control comes from the idea that "language understanding is an intensely *predictive* process. As a new sentence in a conversation starts people seem to operate in a bottom-up manner, piecing together fragments of meaning from the first few words until their conceptual *expectations* can come in to play to drive the understanding process in a more top-down manner. These expectations come from the words themselves, or from the conversational context of preceding exchanges." [29] Thus the Concept Comprehension module uses the idea of expectations associated with words to accomplish the understanding process.

"Expectations are associated with an analyzer dictionary definition in a special type of production rule called a request. Associated with a meaning structure built by a word is a set of expectations embodied in the requests indicating how the structure's characteristic roles should be filled. The special feature of these production-rule-like requests is that, in addition to modifying working memory, which contains the facts describing the current state of the understanding process, they can also add or delete requests. Requests, themselves, are activated when the associated word definition is loaded, that is, they are placed in a short-term memory (the C-LIST) of requests to be considered.

In the algorithm used in this control strategy, the request consideration process repeatedly selects a request and evaluates its test part. If the test returns "non-nil", the request is said to have "fired", its action part is evaluated, and the request is removed from request memory. Requests are capable of checking semantic, lexical or contextual features of the run-time environment, and can create or interconnect conceptual structures." [29] "This strategy continues until all requests have either fired, deleted themselves or are deleted by

another request." [29]

In summary, using these tools - the analyzer dictionary definitions, the ERKS representation scheme, and wordsense definitions, this system allows the user to create Conceptual-Dependency-like representations of input sentences. Because of the design of the analyzer dictionary in the form of the test-action pairs, the processing is driven until all subconcepts associated with the words in the sentence are either combined into one overall concept (the ideal case) or until all requests are exhausted.

CCM Module Operation

With the "Enter Sentence:" prompt, the user begins his interfacing with the acquisition system. Each sentence which is understood is then converted into the conceptual-dependency-like form to be passed on to the next module, the knowledge classification/extraction module. Figure 6a shows the result created from the input of the sentence:

"The engine powers the csd."

This sentence deals with the description of a relationship between two physical objects in the electrical system domain. For a more complex example, dealing with a domain-dependent heuristic, such as verification of a fault, the input would be much more complex. Figure 6b shows this input and the result from the concept comprehension module.

CCM Vocabulary Learning

Because of the importance of the vocabulary found in the analyzer dictionary, it was deemed that vocabulary learning should be incorporated into the system's current implementation. Because noun learning is one of the simpler word classes to learn, noun learning has been incorporated into the system. Through a simple analogy, the system can learn concepts similar to an existing concept (e.g. the generator component is analagous to the engine.) However, because verb learning is more complex, this capability has not yet been included.

4.3 The Knowledge Classification/Extraction Module

At this point, the third system module (KCEM) classifies the input into one of two basic groups; KCEM returns either cs-action (control strategy action) or kb-object (knowl-

edge base object) as a classification response. This classification indicates to the system how to handle the operationalization [35] of the input. Currently the classification strategy is a simple one based on the central primitive of the input. For example, in Figure 6a, the primitive is power. This would be classified as a kb-object indicating that this primitive should have physical objects associated with it. After classifying the input, the appropriate information is extracted using meta-knowledge contained in the domain concept dictionary. At this point, the result of the extraction process is passed to the knowledge incorporation module.

4.4 The Knowledge Incorporation Module

It is this final module KIM, which creates the machine usable form–actual LISP objects or LISP functions, handles the integration of knowledge into new or existing LISP objects, and provides a limited means for detecting and dealing with conflicting information. Receiving information from the extraction process, the knowledge incorporation module, using the domain concept dictionary, converts this information into the appropriate LISP forms. For LISP objects, knowledge integration is simply checking for the existence of a referenced object. If the object is unknown, it is created and the information is added to this new object. If the object already exists, only the new information is added in the appropriate place. In the case of conflicting information, currently implemented only for LISP objects, the user is given the option on what information is to finally be included. For example, if the user first states that **The engine powers the csd** and then states that **The engine powers the generator**, this will be detected as a conflict and the user is given an opportunity to resolve it.

Figure 7a and 7b show the results of this module for examples shown in Figures 6a and 6b respectively. As mentioned earlier, critical to this creation process is the meta-knowledge found in the domain concept dictionary which is discussed next.

The Domain Concept Dictionary

Critical to both the extraction and operationalization processes is the information found in the domain concept dictionary. This is a flavor structure called dcd with instance variables representing meta-knowledge necessary for both the creation of LISP frames and LISP functions. The instance variables include the following: 1) rolelist - a list of the role-names in the particular primitive, 2) path-frame-mapping - the relationship between a path

in the CCM output structure and a slot-value in a created frame, 3) functional-primitive - a LISP expression representing the associated concept, 4) functional-primitive-argument - the argument atom found in the functional-primitive, 5) function-demon - the name of an associated intelligent function used to process information associated with the primitive concept, and 6) path-function-mapping - the relationship between a path in the CCM output and its use in a LISP function definition. The following is a brief overview of how this meta-knowledge is used by AKAS.

In the construction of domain objects, both the role-list and the path-frame-mapping are used. Briefly the role-list is used to generate the access paths to the domain objects in the CCM output. This access permits creation of instances of the mentioned object, the number of instances being controlled by the type of aircraft. The path-frame-mapping tells the system how to establish the relationships between the extracted information and the LISP frame slots.

In the construction of LISP functions (representing heuristic knowledge) the information associated with the instance variables, functional-primitive, functional-primitive-argument, function-demon, and path-function-mapping, are used. The functional primitive provides a template LISP expression which can be used in the creation of a legitimate function. The functional-primitive-argument aids in the substitution process when the appropriate value found in the extracted information is substituted into the LISP expression. The function-demon provides the method to process specific information about a particular structure and the path-function-mapping describes where necessary function elements, such as the function name and lambda list, may be found. Figure 8 shows example entries in the domain concept dictionary.

Explanation Facilities

The system includes the ability to explain the origin of a created frame. By indicating the particular frame object name (e.g., csd1), it is possible to have the system indicate the original input sentence from which it was generated. The names of frames currently existing in the system can be obtained from a system command showing created objects.

It is also possible to obtain the original input source for functions. By entering the function's name, the system will return the original sentence or group of sentences used for the creation of the function.

5. Further Research

As with any system, work on the improvement of fundamental concepts and system features can be an ongoing process. In the case of AKAS, further improvements would be necessary to facilitate its use by a naive computer user. One major area would be the communication between the system and the user. The mixed-initiative dialogue, as advocated in the KLAUS philosophy, needs further improvement in telling the user what sentences are understood and which ones are not. In the course of AKAS' development, the viable sentences were readily known by the authors because these sentences were the desired test cases. Because a naive user would not have this prior information, more guidance would be needed to direct the user in entering sentences that are understood. Perhaps this could be accomplished by some type of preprocessor, in which the input sentence could be scanned with the system determining whether the word order of the sentence is understood or not and telling the user what is not understood. Also, the limited aspects of the vocabulary learning, mentioned above, could be expanded in order to lessen the amount of unknown concepts, thus easing the burden on the user.

Another important aspect of the design, not yet implemented, would be to allow the user to undo concepts he has previously created. This idea is advocated in the KLAUS philosophy and is worthy of inclusion into a viable system. This feature could be accomplished through the use of data dependencies. Establishing dependencies between concepts as they are referenced would permit the linking of information as it becomes available. If a piece of information is removed, then all elements dependent on that information could also be removed.

Because the vocabulary used in the concept comprehension module is critical and complex, it would be worthwhile to create a user interface to aid solely in the construction of the vocabulary. This tool would make it possible to create a seed vocabulary for use in the domain of choice. Coupled with an expanded ability to learn domain concepts, the concept comprehension module could become a very powerful system in any domain context.

6. Conclusion

In this report we have presented the design and implementation of a prototype of the system, called AKAS, for the automatic acquisition of domain and procedural knowledge.

While obviously not solving all possible problems related to automatic knowledge acquisition, the overall system design of AKAS allows the system to work beyond the domain under which it was designed. The separation of the concept comprehension module control strategy from the vocabulary allows its use in any domain. The modularity of the other components of the system could permit a knowledge engineer to modify the routines to recognize the elements of choice for another domain and create the desired machine usable forms.

As indicated in the section on further research there are areas in which this system could be strengthened. This research, along with the other knowledge acquisition research efforts, points out that the knowledge acquisition process continues to be a difficult one to readily and easily capture. This report has offered yet another approach to this problem of capturing knowledge.

D. MACHINE LEARNING OF FAULT CHARACTERISTICS FROM DATA

1. Introduction

In many areas, people are confronted with too many charts, curves, pictures and numerical data coming from experiments, simulations, observations and so on. People are required to find concepts, relationships, descriptions or conclusions from the analysis of data. This process of generating more general and more abstract knowledge (which is closer to the understanding of the problem by human beings) from a lot of individual, specific and lower level information is known as learning from examples.

Because of the great amount of information that needs to be examined, this learning process is complex and very inefficient for a human being to perform. A machine learning system is a good approach in solving this kind of problem. Not only can a machine learning system bring higher efficiency, but it can also accumulate the knowledge for further learning.

We have developed a prototype of a machine learning system, MLS, which takes as input the numerical data (pertaining to normal and faulty jet engine behavior) obtained from an engine simulator, and which generates the characteristic and discriminant descriptions about each kind of fault. These descriptions can be used directly by human experts for understanding the features of each kind of fault and relationships among many sensor parameters, and can be stored in a knowledge base for the future fault diagnosis by an expert system.

2. Forms of Input Data

The jet engine raw data for MLS was obtained from an engine simulator and it contains four parameters with values varying with time. The parameters considered are N1 and N2, which are rotational speeds of low and high pressure turbine assemblies; EGT, which is the exhaust gas temperature; and COMBT, which is the combustion temperature. The first derivative and the second derivative of these parameters were computed at appropriate intervals by the system. The data can be thought of as a curve with the parameter vs. time. This kind of raw data was first passed through a preprocessor which divides each curve into several segments. Each segment is called an event which represents the variation of a parameter in a certain direction. The starting time (or ending time) of the event is

a point where the curve changes its direction. Within each event, the value of the first derivative is either positive, negative or zero. The preprocessor computes the average first derivative for each event, because it was observed that the average value is more meaningful than the individual values along the time axis. The preprocessor also finds the curvature feature from the second derivative for every segment.

The preprocessed data is then passed through a transform program. The output of the transform is the data in relational statement form. Data of two kinds of faults, FUEL INTERRUPTION and BEARING FAILURE, are used for jet engine tests. For each fault there are several examples. Each example consists of four curves for the four parameters (N1, N2, EGT and COMBT). The relational statement form for each example is a conjunction of selectors. Each selector is of the form:

(DESCRIPTOR-i (arg1 ... argn) = value) where arg1, ..., argn are arguments for DESCRIPTOR-i. It is possible for a descriptor to have no arguments. Examples of selectors are shown below:

(N1-START-TIME (N1-event1) = 1.0)
(N2-END-TIME (N2-event2) = 9.6)
(EGT-DURATION (EGT-event1) = short)
(COMBT-DERIVATIVE1 (COMBT-event1) = very-small)
(N1-DIRECTION (N1-event2) = rising)

Notice that except for the time instances all other numerical values are changed to qualitative values for the following reasons:

(1) Easy to use by symbolic manipulations;
(2) Effective for comparison and matching;
(3) Easier to be understood by a human expert.

Significant emphasis has been put on the selection of descriptors which represent important features of the engine faults. Domain knowledge finds its use in the selection process. Unimportant features should be avoided in order to generate good descriptions and to make the learning process more efficient.

The transformation is one kind of generalization, that is, from a range of individual values to a general descriptive word or phrase. One thing needed to take care is that no

significant information should be lost while transforming. Our transform program also uses the constructive generalization rules to add some more selectors which were not in the original descriptions. For instance:

(N1-EVENT-NUMBER(-) = 2)
(CURVE-FEATURE (EGT) = monotonic-rising)

When more than two classes of examples are used as input, as in the rocket engine tests where five types of faults, CONTROL, HPFTP1, HPFTP2, INJECTOR and VALVE faults, are involved, each time one class is chosen as positive examples and the rest classes are taken as negative examples.

The rocket engine raw data was obtained from SSME ground tests, it involved several new parameters. However, descriptors were created in the same way as in the jet engine fault learning.

3. The MLS Learning Technique

3.1 Type of Learning in MLS

Human beings have the ability to make accurate generalizations from individual facts and to discover patterns in observations. This ability is important in knowledge engineering. Since knowledge acquisition is a weak link in the chain of developing knowledge-based systems and since the learning process for a human being is a time-consuming process, simulation of this human ability will enhance machine learning and its applications.

The type of learning used in MLS is a kind of inductive learning. Given a set of specific examples of a concept (positive examples) and a set of examples which are not related to the concept (negative examples), MLS tries to learn the characteristic and discriminant features of the concept and provides symbolic descriptions expressed in general human-oriented terms and forms. In our specific domain the concepts are different engine faults.

3.2 Generalization of Descriptions

Generalization of descriptions is a basic operation in machine learning. The definition of generalization used in the MLS algorithm is described below:

Let p1 and p2 be two products (i.e., conjunctions) of selectors. p1 is a generalization of p2, iff any example e (which is a product of selectors) satisfying p2 implies e is also satisfying p1. Similar definition of generalization and some related concepts can be found in [2-4].

Example 1.

Let

p1 = (N1-DIRECTION (N1-event1) = falling)

p2 = (N1-DIRECTION (N1-event1) = falling) (N1-DURATION (N1-event1) = large)

then any e satisfying p2 implies e satisfying p1. So p1 is a generalization of p2.

Example 2.

Let

p1 = (CURVE-FEATURE (EGT) = monotonic)

p2 = (EGT-EVENT-NUMBER (-) = 1)

then any e satisfying p2 implies e is satisfying p1. So p1 is a generalization of p2.

The generalization concept is defined with a different perspective below:

Let s1 = $\{e|p1\}$ be the set of all possible instances satisfying p1, and s2 = $\{e|p2\}$. p1 is a generalization of p2, iff s2 is a subset of s1.

3.3 Covering

The MLS algorithm employs covering for testing the discriminating power of a description. The definition of covering, as used in the algorithm, is given below:

A description D is a disjunction of one or more products of selectors. When we say that D covers an example e1, we mean that there is a product p in D such that e1 $\in s = \{e|p\}$, i.e., e1 satisfies p.

3.4 The Learning Algorithm of MLS

The inductive learning method used in MLS is similar to the STAR methodology described by Michalski [36]. The following features were created to meet the domain requirements in the development of MLS:

(1) MLS generates characteristic descriptions as well as discriminant descriptions.

(2) MLS makes full use of the STAR generated from an example, instead of using only one element of it. This improves the efficiency significantly.
(3) More than one disjunction can be generated for both kinds of descriptions. Therefore, more descriptions can be found.
(4) More than two classes of examples can be used as input, which generalizes learning within a larger context.
(5) Domain knowledge, stored in MLS' knowledge base, is employed to select descriptions which best discriminate faults. Description selection based on this knowledge makes it possible to accumulate learned concepts for further learning.

The following is an outline of the learning algorithm:

Let POS and NEG be sets of selector products representing positive and negative examples of a concept, respectively.

(1) read in data to POS and NEG;
(2) apply construction generalization rules to POS and NEG;
(3) randomly select an example e from POS;
(4) maximally generalize e in various ways by using the dropping-condition generalization rule to create a list of generalized descriptions GD;
(5) sort the descriptions in GD from the most to the least preferred descriptions according to the preference CRITERION1 described later;
(6) test each element in GD for consistency and completeness, collect those satisfying certain conditions or thresholds in lists: SOLUTIONS, COMPLETE, CONSIST and CHARACTER, respectively;
(7) if the size of SOLUTIONS or the size of CONSIST is large enough, then go to step (10);
(8) specialize each description d in GD in various ways by appending to it an element from the original GD which is of lower preference than d or lower than the last element appended to d; then sort GD and retain a limited number of elements;
(9) if no d in GD can be specialized further, then go to step (10), otherwise go to step (6);
(10) collect SOLUTIONS into DIS-DES, a list of discriminant descriptions; collect COMPLETE into CHA-DES, a list of characteristic descriptions;
(11) sort CONSIST according to the preference CRITERION2; then append the first m elements of CONSIST to CONSISTENT;

(12) if a minimum number of discriminant descriptions have not been generated and the size of CONSISTENT is not large enough, then select a different e from POS and go to step (4); on the other hand, if all the examples in POS have been tested, then go to the next step;

(13) process CONSISTENT to get more discriminant descriptions and process CHARACTER to get more characteristic descriptions;

(14) sort DIS-DES by CRITERION2 and sort CHA-DES by CRITERION1;

(15) insert the most preferred descriptions of the concept into the knowledge base and display the results.

3.5 Preference Criteria

CRITERION1:

In the MLS algorithm described in the previous section, CRITERION1 is used to choose the promising generalized descriptions. Two tolerances, TAU1 and TAU2, are used to delete unpromising descriptions. The negative covering number (the number of negative examples covered by a description) of each description d in GD should be less than TAU1 and the positive covering number of d should be greater than TAU2. The values of TAU1 and TAU2 are determined by the number of input examples and the requirement of minimizing the computing time. The preference value function is:

$$PV1(d) = poscov(d) \times w1 - negcov(d) \times w2 + w(d)$$

where poscov(d) and negcov(d) are positive and negative covering numbers of d, w1 and w2 are weights, and w(d) is the sum of weights of all the descriptors in d. A larger preference value means a description is more promising. We prefer those descriptions with a larger positive covering number and a smaller negative covering number. Usually w2 is given a larger value because a description covering 50 percent of positive examples can be considered as promising but a description covering 50 percent of negative examples is definitely unpromising, so negcov is more important than poscov. The more important a descriptor is, the larger value will be given to its weight. The purpose of this is to emphasize important features.

CRITERION2:

It is preferable to find fewer discriminant descriptors to make the differentiation among engine faults. CRITERION2 was designed for this purpose. It is used to sort CONSIST

and DIS-DES. Since the neg-cov of every description in these lists is zero, the preference value function is:

PV2(d) = poscov(d) × w1 + w(d) - N(d) where N(d) is the number of descriptors in d. A smaller value of N is preferred. Poscov(d) and w(d) have the same meanings as those in the CRITERION1.

3.6 Generalization Rules

Several generalization rules have been implemented in MLS. A few examples of these rules are described below:

(a) Climbing-generalization-tree rule: This rule is applicable only to descriptions involving clustering hierarchical descriptors. For example:

For the domain of jet engine faults, this rule is used to convert numerical data within a certain range into a descriptive word or phrase. For instance, if the duration of an event in the N1 parameter for some fault is less than 0.2 second, then the selector

(N1-DURATION (N1-event1) = very-short)

will be generated.

(b) Closing-the-interval rule: If a kind of fault is described by two expressions e1 and e2, where e1 contains a selector

(N1-EVENT-NUMBER(-) = 2),

and e2 contains a selector

(N1-EVENT-NUMBER(-)=3),

then they can be replaced by the selector

(N1-EVENT-NUMBER(-) = 2..3).

Here the descriptor N1-EVENT-NUMBER should be denoted as a linear descriptor.

(c) Counting-quantified-variable rule: This is a constructive generalization rule used to add more selectors to a description. For instance, if an example description p contains several selectors:

(EGT-EVENT (e1)),

...,

(EGT-EVENT (e5)),

then the selector

(EGT-EVENT-NUMBER(-) = 5)

will be added to p, which describes the total number of EGT parameter events in this example.

(d) Counting-relation-argument rule: If a description d about a concept contains a selector
(REL (X1 X2 X3)),
which states that X1, X2, and X3 have some relation, then a new selector
(REL-NUMBER = 3)
will be constructed and added to d, which counts the number of arguments in the relation REL.

(e) Generating-chain-property rule: If a descriptor represents a transitive relationship, then a new selector can be constructed to describe the minimum and maximum elements on the transitive chain. For instance, with description
[(BEFORE(N1-event1 N2-event1)) (BEFORE (N2-event1 EGT-event1))],
new selectors
(FIRST-BEFORE=N1-event1) and
(LAST-BEFORE=EGT-event1)
can be constructed and added to the description.

3.7 Guiding the Learning Process with Domain Knowledge

To describe a concept, we use many descriptors to represent different features of the concept. Obviously, some features are more important, while others are less important. Taking our flight engine fault domain as an example, the change of rate of a parameter is more important than the starting time of the change. We also know that some parameters would change more significantly in one type of fault than in other types of faults. Therefore, in the learning process we need to pay more attention to the more significant parameters. EGT and COMBT are more sensitive to engine faults caused by FUEL INTERRUPTION. So in the learning process with input examples related to FUEL INTERRUPTION, we give a higher preference value to the descriptors: EGT-DERIVATIVE-1, COMBT-DERIVATIVE-1, EGT-DIRECTION and COMBT-DIRECTION.

Domain knowledge also indicates that a BEARING FAILURE would cause N1 and N2 to decrease slowly and steadily. This means that the first derivatives of N1 and N2 would be small and the second derivative of them would be zero. Therefore, we can assign a higher preference value to descriptors: N1-DERIVATIVE-1, N1-CURVATURE, N2-DERIVATIVE-1 and N2-CURVATURE.

These assignments of preference values, employing domain knowledge, cause MLS to concentrate on the key features of each kind of engine fault. This will improve the computation efficiency as well as the usefulness of the resulting descriptions.

4. Generated Descriptions of a Concept

The output of MLS is *(i)* a list of discriminant descriptions and *(ii)* a list of characteristic descriptions about a concept which in our domain is a type of engine fault. The discriminant descriptions, which describe all the features possessed by a specific engine fault and not by any other faults, are used to differentiate possible engine faults. The characteristic descriptions, which describe all the features appearing in the examples of a specific engine fault, are used to characterize the given fault examples.

Both kinds of descriptions are in the disjunctive form. The number of products in a disjunction is limited to at most three for the purposes of easy understanding and high efficiency of computation.

These descriptions are available as output in a print format and are also stored in a knowledge base. The descriptions stored in the knowledge base can be used as domain knowledge in the engine fault diagnosis later.

The MLS results for jet engine described in this paper include only two types of engine faults: FUEL INTERRUPTION and bearing failure. Examples of output are shown below.

The discriminant descriptions of engine faults caused by FUEL INTERRUPTION are:

(COMBT-DIRECTION (COMBT-event1) = falling)

(CURVE-FEATURE (EGT) = non-monotonic)

(OR (COMBT-EVENT-NUMBER (-) = 2)

(COMBT-DERIVATIVE-1 (COMBT-event1) = large))

The characteristic descriptions of these engine faults caused by FUEL INTERRUPTION are:

(N2-DIRECTION (N2-event1) = falling)

(CURVE-FEATURE (N2) = monotonic-falling)

(OR(N1-DURATION (N1-event1) = long) (EGT-DURATION (EGT-event2) = short)

Among the output descriptions some may be redundant, some may be easily derived

from other descriptions, and some combinations of features may be more preferable than others. Thus, from the generated descriptions, we need to select a set of the most effective ones by employing domain knowledge. For example, the domain knowledge shows that the combustion temperature (COMBT), dominates the exhaust gas temperature (EGT), which means that whenever COMBT drops drastically, EGT also drops. In our knowledge base this is represented by the following rule:

RULE12:

(IF (AND (COMBT-DIRECTION (?EVENT) = falling)
(COMBT-DERIVATIVE-1 (?EVENT) = large))
THEN (EGT-DIRECTION (?EVENT) = falling)

Suppose in our result there are two descriptions,

D1:

(AND (COMBT-DIRECTION (COMBT-event1) = falling)
(COMBT-DERIVATIVE-1 (COMBT-event1) = Large))

D2:

(EGT-DIRECTION (EGT-event1) = falling)

the second description is redundant and should not be stored in the knowledge base.

In another example, suppose there are two descriptions,

D1:

(AND (COMBT-DIRECTION (COMBT-event1) = falling)
(COMBT-DERIVATIVE-1 (COMBT-event1) = large))

D3:

(EGT-DIRECTION (EGT-event1) = rising)

our learning program would deduce that a sensor failure has happened. In this case all descriptors involving the sensor would be canceled.

5. Conclusion

MLS has been implemented on a Symbolic 3670 Lisp machine using COMMON LISP. Data of jet engine faults generated by an engine simulator was used to test MLS. The results have shown that MLS can help to conceptualize different types of engine faults.

The process of developing MLS has shown that domain knowledge is very important in the design of a task-oriented learning system. Without any constraints and guidance based on domain knowledge, the learning system would fail to generate effective descriptions in

reasonable time.

We feel that more improvements are still needed in MLS, such as *(i)* including more types of generalization and specialization rules, *(ii)* adding more domain knowledge for the guidance of learning, *(iii)* choosing descriptions based on the relationship of selectors and their most desired capabilities, and *(iv)* adding incremental learning in which partially learned concepts can be used for further learning.

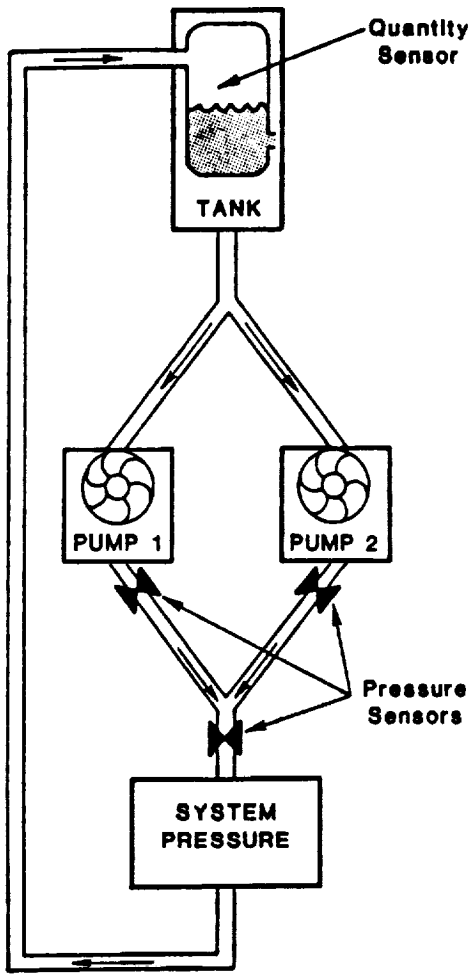


Figure 1. A simplified hydraulic system.

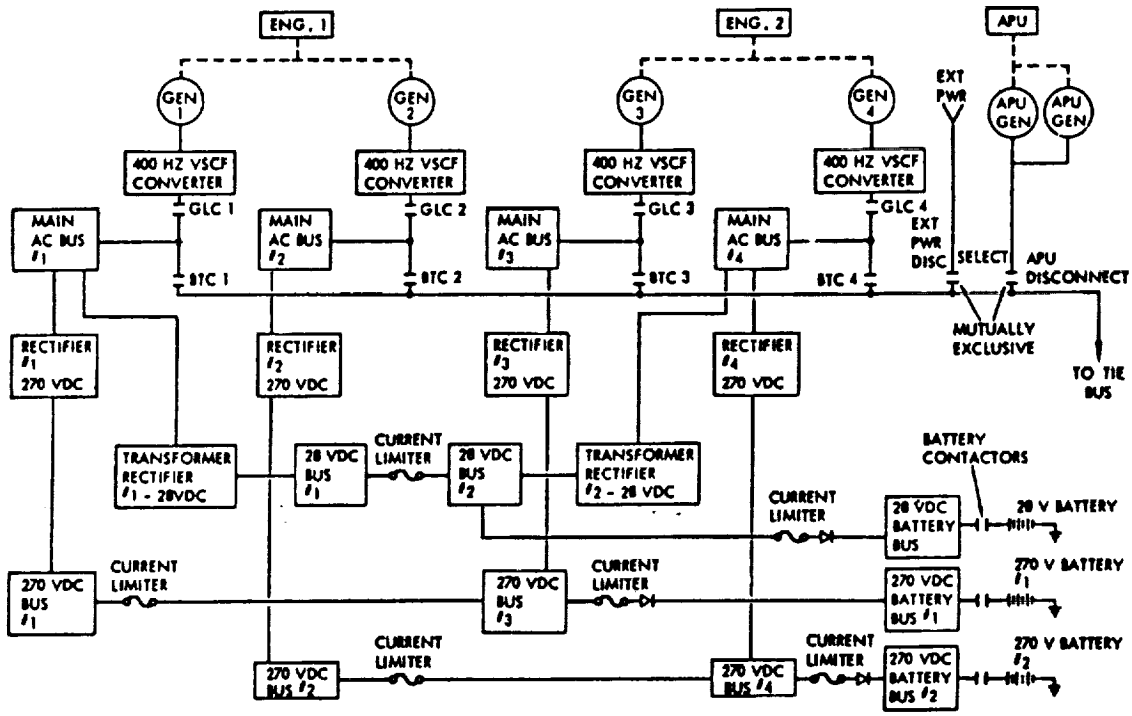


Figure 2. Main bus distribution for the Lockheed Advanced Concept System airplane. Permission to reprint from Sexton [9] granted by the NASA Langley Research Center.

(frame gen #1-freq
(value = 393)
(range = 380–420)
(associated fault = generator #1)]

(frame gen #1-ac-volt
(value = 118)
(range = 110–130)
(associated fault = generator #1)]

Figure 3. Examples of Sensor Frames

(frame generator #1
(status = on)
(degree of danger = 70)
(likelihood = 60)
(parent = engine #1)
(children = vscf #1)
(associated sensors = gen #1-freq, gen #1-ac-volt)]

(frame vscf #1
(status = on)
(degree of danger = 60)
(likelihood = 30)
(parent = generator #1)
(children = ac bus #1)
(associated sensors = vscf #1-freq, vscf #1-ac-volt)]

Figure 4. Examples of Fault Frames

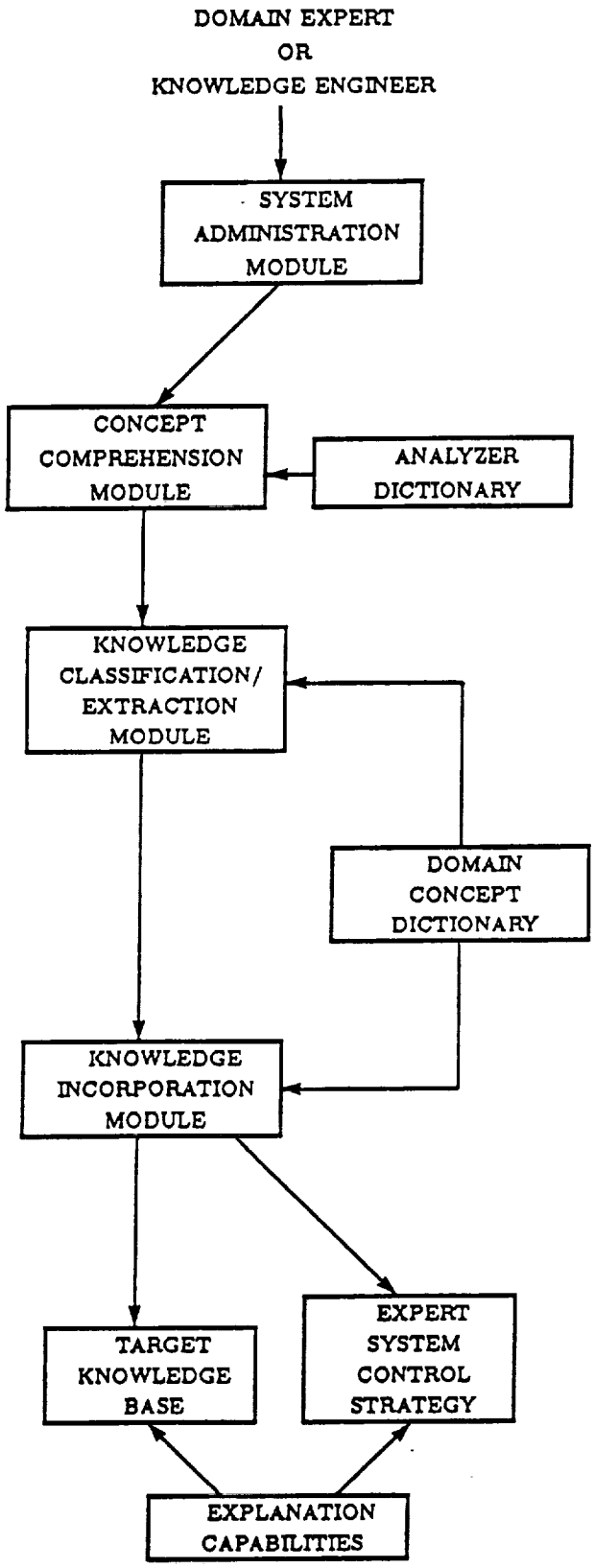


Figure 5 AKAS Architecture

CCM Input:

The engine powers the csd.

CCM Output/KCEM Input:

```
(POWER
   POWER-SOURCE
(ELECTRICAL-COMPONENT
            REF (DEF)
            COMP-NAME (ENGINE)
            PARTOF (ELECTRICAL-SYSTEM))
   POWERED-OBJECT
(ELECTRICAL-COMPONENT
            REF (DEF)
            COMP-NAME (CSD)
            PARTOF (ELECTRICAL-SYSTEM)))
```

KCEM Output/KIM Input:

Classification:

KB-OBJECT

Extracted Information:

```
(FRAME-COMPONENTS (ENGINE) (CSD))
(RELATIONSHIP (PARENTS (ENGINE) (CSD)))
(RELATIONSHIP (CHILDREN (CSD) (ENGINE)))
```

Figure 6a. Knowledge Base Input Example

CCM Input:

TO VERIFY A FAULT HYPOTHESIS SEE IF
THERE IS MORE THAN ONE SUPPORTING
SENSOR. IF THERE IS MORE THAN ONE
SUPPORTING SENSOR THEN THE FAULT
IS VERIFIED.

CCM Output/KCEM Input:

```
(DETERMINE
   PURPOSE
(VERIFY VERIFIED-STATE
(MALFUNCTION
            REF (INDEF)
            OBJECT-CLASS (CURRENT)
            M-STATE-VALUE (VERIFIED)))
   METHOD
(ACTIONS ACTION-1
(IF-TEST ANTECEDENT
    (MORE-THAN-ONE
        OBJECT (SUPPORTING-SENSOR))
        CONSEQUENT
    (MALFUNCTION OBJECT (CURRENT)
         STATE-VALUE (VERIFIED)))))
```

KCEM Output/KIM Input:

Classification:

CS-ACTION

Extracted Information:

```
(FUNCTION-NAME VERIFY)
(FUNCTION-LAMBDA-LIST (CURRENT))
(FUNCTION-BODY (ACTIONS ACTION-1
(IF-TEST ANTECEDENT
    (MORE-THAN-ONE
        OBJECT (SUPPORTING-SENSOR))
        CONSEQUENT
    (MALFUNCTION OBJECT (CURRENT)
        STATE-VALUE (VERIFIED)))))
```

Figure 6b. Control Strategy Input Example

```
(FRAME ENGINE1
    IS A COMPONENT
    WITH
    (SENSORS = ())
    (CHILDREN = (CSD1))
    (PARENTS = ())
    (DEGREE-OF-DANGER = ())
    (FAULT-RATE = ())


(FRAME CSD1
    IS A SENSOR
    WITH (SENSORS = ())
    (CHILDREN = ())
    (PARENTS = (ENGINE1))
    (DEGREE-OF-DANGER = ())
    (FAULT-RATE = ())
```

a) Lisp objects created from example 2a

```
(DEFUN VERIFY (CURRENT)
    (cond ((not(null(cdr supporting-sensor))) t)))
```

b) Lisp function created from example 2b

Figure 7 Created Lisp Objects

```
(setf power
     (make-instance 'dcd
          :role-list '(power-source powered-object)
          :path-frame-mapping
          '((power-source parents)
          (powered-object children))))

(setf determine
     (make-instance 'dcd
          :role-list '(purpose method)
          :path-function-mapping
          '((purpose (name lambda-list))
          (method (function-body)))))

(setf more-than-one
     (make-instance 'dcd
          :role-list '(object)
          :functional-primitive
          '(not (null (cdr more-than-one-arg)))
          :functional-primitive-argument
          'more-than-one-arg
          :function-demon 'more-than-one-demon))
```

Figure 8 Domain Concept Dictionary Entries

E. REFERENCES

[1] Product Safety Group

"Statistical Summary of Commercial Jet Aircraft Accidents, World Wide Operations, 1959–1980," Boeing Commercial Airplane Company, Seattle, Washington.

[2] Hanson, D. C., Howison, W. W. , Chikos, S. F. , and Berson, B. L.

"Aircraft Alerting Systems Standardization Study, Phase IV, Accident Implications on Systems Design," Final Report No. DOT/FAA/RD–82/26, June, 1982, U.S. Dept. of Transportation, Federal Aviation Administration Systems Research and Development Service, Washington, D.C.

[3] Gravely, M. L., and Mysing, J. O.

"The Influence of Smart Computers on the Cockpit of the Future," Proceedings of the IEEE 1981 National Aerospace and Electronics Conference, Dayton, Ohio, May, 1981, vol. 2, pp. 549–556.

[4] Clay, C. W.

"Digital Electronic Flight Decks: The Outlook for Commercial Aviation," IEEE Trans. on Aerospace and Electronic Systems, vol. 20, no. 2, March, 1984, pp. 221–226.

[5] Smead, F. W.

"Voice–Actuated Avionics," Astronautics and Aeronautics, vol. 21, no. 10, October, 1983, pp. 54–63.

[6] Anderson, B. M., Cramer, M. L. , Lystad, G. S. and Stern, R. C.

"Intelligent Automation of Emergency Procedures in Advanced Fighter Aircraft," Proceedings of the IEEE 1st Conference on Artificial Intelligence Applications, Denver, Colorado, December, 1984, pp. 496–501.

[7] Cross, S. E.

"Towards and Expert System Architecture for Flight Domain Applications," Proceedings of the IEEE 1984 National Aerospace and Electronics Conference, Dayton, Ohio, Mary, 1984, vol. 2, pp. 784–788.

[8] Klos, L. C., Edwards, J. A. and Davis, J. A.

"Artificial Intelligence - - An Implementation Approach for Advanced Avionics," Proceedings of the AIAA Computers in Aerospace IV Conference, Hartford, Conn., October, 1983, pp. 300–307.

[9] Minsky, M.
"A Framework for Representing Knowledge," in P. Winston (Ed.), The Psychology of Computer Vision, McGraw–Hill: New York, 1975, pp. 211–217.

[10] Erman, L. D., Hayes-Roth, F. , Lesser, V. R. and Reddy, D. R.
"The Hearsay–II Speech–Understanding System: Integrating Knowledge to Resolve Uncertainty," ACM Computing Surveys, vol. 12, no. 2, June, 1980, pp. 213–252.

[11] Doyle, J.
"Truth Maintenance Systems for Problem Solving," MIT AI Laboratory Technical Report No. 419, January, 1978.

[12] Shortliffe, E. H.
Computer-Based Medical Consultations: MYCIN. New York: American Elsevier, 1976.

[13] Ali, M., et. al.
"Knowledge-based Computer Expert Systems" Tabes–85, Huntsville, AL, April 23–25, 1985

[14] Ali, M. et. al.
"On-board Fault Monitoring and Diagnosis in Airplanes" Joint AEDC/UTSI Technical Symposium, May, 1985

[15] Ali, M., Scharnhorst, D. A., AI, C. -S. and Ferber, H. J.
"Expert Systems for Real-Time Fault Monitoring and Diagnosis in Space Vehicles," Conference on Artificial Intelligence: From Space...Down to Earth, Huntsville, AL, October, 1985.

[16] Ali, M. et. al.
"A Flight Expert System (FLES) for On-Board Fault Monitoring and Diagnosis" The International Symposium and Exposition on Industrial Artificial Intelligence Systems, Research, Applications, & Software Development, November 4–7, 1985.

[17] Ali, M. and Scharnhorst, D. A.
"Knowledge-Based Avionics for Interactive Fault Assessment," WATTEC 86: 13th Annual Energy Conference, Knoxville, TN February, 1986.

[18] Ali, M. and Scharnhorst, D. A.
"Sensor-Based Fault Diagnosis in a Flight Expert System," in Proceedings of IEEE 2nd Conference on Artificial Intelligence Applications, Miami, FL, December, 1985.

[19] Ali, Scharnhorst, D. A. , Ai, C. -S. and Ferber, H. J.
"A Flight Expert System (FLES) for On-board Fault Monitoring and Diagnosis," Proceedings of SPIE Technical Symposium on The Applications of Artificial Intelligence III, Orlando, FL, April, 1986.

[20] Ferber, H. J. and Ali, M.
"Automatic Acquisition of Domain and Procedural Knowledge," Proceedings of The First International Conference on Industrial and Engineering Applications of Artificial Intelligence & Expert Systems" pp. 762–771, June 1–3, 1988.

[21] Ferber, H. J.
"Automating the Knowledge Acquisition Process in Expert System Development," Master's Thesis, UTSI, June, 1988.

[22] Bruce, B. C.
"A Model for Temporal References and its Application in a Question Answering Program," Artificial Intelligence, 3, 1972.

[23] Kahn, K. M. and Gorry, G. A.
"Mechanizing Temporal Knowledge," Artificial Intelligence, 9, 1977.

[24] McDermott, D.
"A Temporal Logic for Reasoning About Processes and Plans," Research Report 196, Dept. of Computer Science, Yale University 1981.

[25] Allen, J. F.
"An Interval-based Representation of Temporal Knowledge," Proceedings of the 7th IJCAI, Vancouver, B.C., August, 1981.

[26] Allen, J. F. and Kooman, J. A.
"Planning Using a Temporal World Model," Proceedings of the 8th IJCAI, Karlsruhe, West Germany, 1983.

[27] Abbott, K. H., Schutte, P., Palmer, M., and Ricks, W.
"Fault finder: a diagnostic expert system with graceful degradation for onboard aircraft applications" 14th International Symposium on Aircraft Integrated Monitoring Systems, Friedrichshafen, West Germany, September, 1987.

[28] Abbott, K. H.
"Robust Fault Diagnosis of Physical Systems in Operation" Ph.D. Dissertation, Rutgers University, May, 1990.

[29] Cullingford, Richard E.
Natural Language Processing: A Knowledge Engineering Approach, Totowa, NJ: Rowman and Littlefield, 1986.

[30] David, R. and Lenat, D. B.
Knowledge-Based Systems in Artificial Intelligence, New York: McGraw-Hill International Book Company, 1982.

[31] Fiegenbaum, E. A.
Knowledge Engineering:The Applied Side of Artificial Intelligence, Defense Technical Information Center Technical Report AD A092574, Stanford Heuristics Programming Project Report No. HPP-80-21, Department of Computer Science, Stanford University Report No. STAN-CS-80-812, September, 1980.

[32] Haas, N and Hendrix, G. G.
"Learning By Being Told: Acquiring Knowledge for Information Management." Machine Learning: An Artificial Intelligence Approach. Michalski, Ryszard S., Jaime G. Carbonell, and Tom M. Mitchell, eds., Los Altos, CA: Morgan Kaufmann Publishers, Inc., 1983.

[33] Hayes-Roth, F., Klahr, P. and Mostow, D. J.
"Knowledge Acquisition, Knowledge Programming, and Knowledge Refinement," Rand Paper R-2540-NSF, The Rand Corporation, May, 1980.

[34] Duda, R. O., et al,
"Development of the PROSPECTOR Consultation system for mineral exploration" Final Report. SRI Projects 5821 and 6415, Artificial Intelligence Center, SRI International, Menlo Park, CA., 1978.

[35] Mostow, J and Hayes-Roth, F.
"Machine-aided Heuristic Programming: A Paradigm for Knowledge Engineering" Rand: Santa Monica, CA, Report number N-1007-NSF, February, 1979.

[36] Michalski, R. S., Carbonell, J. G., and Mitchell, T. M.
"Machine Learning, An Artificial Intelligence Approach," Morgan Kaufmann, Los Altos, 1983.

[37] Vere, S. A.
"Induction of Concepts in the Predicate Calculus," Fourth International Joint Conference on Artificial Intelligence, Tbilisi, USSR, pp. 281–287, 1975.

[38] Vere, S. A.
"Induction of Relational Productions in the Presence of Background Information," Fifth International Joint Conference on Artificial Intelligence, Cambridge, Mass., pp. 349–355.

[39] Vere, S. A.
"Multilevel Counterfactuals for Generalizations of Relational Concepts and Productions," Artificial Intelligence, Vol. 14, No. 2, pp. 138–164, September, 1980.

F. ACKNOWLEDGEMENT

There are many researchers who contributed to this project. Mr. Harry Ferber, Mr. Dean Scharnhorst, Mr. Min Ke and Mr. Lawrence Ai's contributions are particularly notable. We wish to thank domain experts, pilots and mechanics who assisted us in acquiring the needed domain knowledge. We are particularly indebted to Ms. Chris Barrett, Mr. Robert S. Jones, and Captain James C. Kinnard whose expertise made learning about airplanes easier. We also thank Dr. Kathy Abbott and Mr. Sam Morello for many useful comments and technical discussions as well as for supporting this project.

G. PAPERS AND PRESENTATIONS

1. Ali, M., et. al.
   "Knowledge-based Computer Expert Systems" Tabes–85, Huntsville, AL, April 23–25, 1985

2. Ali, M. et. al.
   "On-board Fault Monitoring and Diagnosis in Airplanes" Joint AEDC/UTSI Technical Symposium, May, 1985

3. Ali, M., Scharnhorst, D. A., AI, C. -S. and Ferber, H. J.
   "Expert Systems for Real-Time Fault Monitoring and Diagnosis in Space Vehicles," Conference on Artificial Intelligence: From Space...Down to Earth, Huntsville, AL, October, 1985.

4. Ali, M. et. al.
   "A Flight Expert System (FLES) for On-Board Fault Monitoring and Diagnosis" The International Symposium and Exposition on Industrial Artificial Intelligence Systems, Research, Applications, & Software Development, November 4–7, 1985.

5. Ali, M. and Scharnhorst, D. A.
   "Knowledge-Based Avionics for Interactive Fault Assessment," WATTEC 86: 13th Annual Energy Conference, Knoxville, TN February, 1986.

6. Ali, M. and Scharnhorst, D. A.
   "Sensor-Based Fault Diagnosis in a Flight Expert System," in Proceedings of IEEE 2nd Conference on Artificial Intelligence Applications, Miami, FL, December, 1985.

7. Ali, Scharnhorst, D. A. , Ai, C. -S. and Ferber, H. J.
   "A Flight Expert System (FLES) for On-board Fault Monitoring and Diagnosis," Proceedings of SPIE Technical Symposium on The Applications of Artificial Intelligence III, Orlando, FL, April, 1986.

8. Ferber, H. J. and Ali, M.
   "Automatic Acquisition of Domain and Procedural Knowledge," Proceedings of The First International Conference on Industrial and Engineering Applications of Artificial Intelligence & Expert Systems" pp. 762–771, June 1–3, 1988.